Java CyclicBarrier: Example Application

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of Java CyclicBarrier
- Recognize the key methods in the Java CyclicBarrier
- Know how to program with Java CyclicBarrier in practice

class GCDCyclicBarrierWorker implements Runnable {
    private final CyclicBarrier mEntryBarrier;
    private final CyclicBarrier mExitBarrier; ...

    GCDCyclicBarrierWorker(CyclicBarrier entryBarrier,
                            CyclicBarrier exitBarrier, ...) {
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;
        ...
    }

    public void run() {
        ...
        mEntryBarrier.await();
        runTest();
        mExitBarrier.await();
        ...
    }
}
Overview of the GCD App
Overview of the GCD App

• This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms

Overview of the GCD App

- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms.
- GCD computes the largest positive integer that is a divisor of two numbers.
  - e.g., the GCD of 80 & 120 = 40

See [en.wikipedia.org/wiki/Greatest_common_divisor](en.wikipedia.org/wiki/Greatest_common_divisor)
Overview of the GCD App

- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
- GCD computes the largest positive integer that is a divisor of two numbers
- Four GCD algorithms are tested
Overview of the GCD App

• This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
  • GCD computes the largest positive integer that is a divisor of two numbers
  • Four GCD algorithms are tested
    • The gcd() method defined by BigInteger

See docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html#gcd
Overview of the GCD App

• This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
  • GCD computes the largest positive integer that is a divisor of two numbers
  • Four GCD algorithms are tested
    • The gcd() method defined by BigInteger
    • An iterative Euclid algorithm

See en.wikipedia.org/wiki/Euclidean_algorithm
Overview of the GCD App

• This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
  • GCD computes the largest positive integer that is a divisor of two numbers
  • Four GCD algorithms are tested
    • The gcd() method defined by BigInteger
    • An iterative Euclid algorithm
    • A recursive Euclid algorithm

Overview of the GCD App

• This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
  • GCD computes the largest positive integer that is a divisor of two numbers
  • Four GCD algorithms are tested
    • The gcd() method defined by BigInteger
    • An iterative Euclid algorithm
    • A recursive Euclid algorithm
    • A complex GCD algorithm that uses binary arithmetic

See [en.wikipedia.org/wiki/Binary_GCD_algorithm](en.wikipedia.org/wiki/Binary_GCD_algorithm)
Overview of the GCD App

- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms.
- GCD computes the largest positive integer that is a divisor of two numbers.
- Four GCD algorithms are tested:
  - The gcd() method defined by BigInteger
  - An iterative Euclid algorithm
  - A recursive Euclid algorithm
  - A complex GCD algorithm that uses binary arithmetic.

However, the details of these algorithms are not important for our discussion.
GCDCyclicBarrierTest
Class Walkthrough
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier = 
            new CyclicBarrier(gcdTests.size() + 1, () -> 
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier = 
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> 
                new Thread(new GCDCyclicBarrierWorker(entryBarrier, exitBarrier, 
                    gcdTuple, this)).start());
            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));

        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                gcdTuple, this)).start());

            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier = new CyclicBarrier(gcdTests.size() + 1, () ->
            GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new GCDCyclicBarrierWorker(entryBarrier,
                exitBarrier, gcdTuple, this)).start());
            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}
• Create worker threads that use exit & entry barrier CyclicBarrier objects

```java
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier = new CyclicBarrier(gcdTests.size() + 1, () ->
            GCDCyclicBarrierWorker.initializeInput(sITERATIONS));

        CyclicBarrier exitBarrier = new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                gcdTuple, this)).start());

            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
```
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));

        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple ->
                new Thread(new GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                    gcdTuple, this)).start());

            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
                new CyclicBarrier(gcdTests.size() + 1, () ->
                                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier = Create exit barrier
                new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple ->
                        new Thread(new GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                                gcdTuple, this)).start());

            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}

We add a "+ 1" for the thread that initializes the tests
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                    gcdTuple, this)).start());

            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}
• Create worker threads that use exit & entry barrier `CyclicBarrier` objects

```java
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                gcdTuple, this)).start());

            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}
```

**Create & start threads w/barriers**
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
            GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                gcdTuple, this)).start());
            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}
GCDCyclicBarrierTest Class Walkthrough

• Create worker threads that use exit & entry barrier CyclicBarrier objects

```java
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                    gcdTuple, this)).start());

            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
```

Let all worker threads proceed at the same time, fixing limitation with CountDownLatch

See previous lesson on "Java CountDownLatch"
**GCDCyclicBarrierTest Class Walkthrough**

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```java
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                gcdTuple, this)).start());
            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}
```

After `await()` returns for a CyclicBarrier it will be reset (& is thus reusable) *without* needing to create a new CyclicBarrier instance
GCDCyclicBarrierWorker
Class Walkthrough
This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation.

class GCDCyclicBarrierWorker implements Runnable {
    private final CyclicBarrier mEntryBarrier;
    private final CyclicBarrier mExitBarrier;
    ...

    GCDCyclicBarrierWorker(CyclicBarrier entryBarrier,
                            CyclicBarrier exitBarrier, ...) {
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;
        ...
    }

    public void run() {
        ...
        mEntryBarrier.await();
        runTest();
        mExitBarrier.await();
        ...
    }
}

See GCD/CyclicBarrier/app/src/main/java/edu/vandy/gcdtesttask/presenter/GCDCyclicBarrierWorker.java
GCDCyclicBarrierWorker Class Walkthrough

- This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

```java
class GCDCyclicBarrierWorker implements Runnable {
    private final CyclicBarrier mEntryBarrier;
    private final CyclicBarrier mExitBarrier;
    ...

    GCDCyclicBarrierWorker(CyclicBarrier entryBarrier, CyclicBarrier exitBarrier, ...) {
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;
        ...
    }

    public void run() {
        ...
        mEntryBarrier.await();
        runTest();
        mExitBarrier.await();
        ...
    }
}
```

Initialize barrier fields
GCDCyclicBarrierWorker Class Walkthrough

- This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

```java
class GCDCyclicBarrierWorker implements Runnable {
    private final CyclicBarrier mEntryBarrier;
    private final CyclicBarrier mExitBarrier;
    ...

    GCDCyclicBarrierWorker(CyclicBarrier entryBarrier,
                            CyclicBarrier exitBarrier, ...) {
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;
        ...
    }

    public void run() {
        ...
        mEntryBarrier.await();
        runTest();
        mExitBarrier.await();
        ...
    }
```

This hook method executes after the thread is started
GCDCyclicBarrierWorker Class Walkthrough

• This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

class GCDCyclicBarrierWorker implements Runnable {
    private final CyclicBarrier mEntryBarrier;
    private final CyclicBarrier mExitBarrier;
    ...

    GCDCyclicBarrierWorker(CyclicBarrier entryBarrier,
                           CyclicBarrier exitBarrier, ...) {
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;
        ...
    }

    public void run() {
        ...
        mEntryBarrier.await();
        runTest();
        mExitBarrier.await();
        ...
    }
}

This entry barrier causes all worker threads to wait until they are all ready, thus fixing the earlier limitation with CountDownLatch

See previous lesson on “Java CountDownLatch”
GCDCyclicBarrierWorker Class Walkthrough

• This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

class GCDCyclicBarrierWorker implements Runnable {
    private final CyclicBarrier mEntryBarrier;
    private final CyclicBarrier mExitBarrier;
    ...

    GCDCyclicBarrierWorker(CyclicBarrier entryBarrier,
                           CyclicBarrier exitBarrier, ...) {
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;
        ...
    }

    public void run() {
        ...
        mEntryBarrier.await();
        runTest(); // Run the GCD algorithm associated with this object
        mExitBarrier.await();
        ...
    }
}
GCDCyclicBarrierWorker Class Walkthrough

- This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

class GCDCyclicBarrierWorker implements Runnable {
    private final CyclicBarrier mEntryBarrier;
    private final CyclicBarrier mExitBarrier;
    ...

    GCDCyclicBarrierWorker(CyclicBarrier entryBarrier,
                           CyclicBarrier exitBarrier, ...) {
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;
        ...
    }

    public void run() {
        ...
        mEntryBarrier.await();
        runTest();
        mExitBarrier.await();
        ...
    }
}

Exit barrier waits until all threads are done before returning
End of Java CyclicBarrier: Example Application